Developing Programming Abstractions for Cyberphysical Systems

Sayan Mitra
mitras@illinois.edu
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign

Instruction Set Architectures (ISAs) define a generic interface of hardware for programming computers. The execution of an ADD instruction, for example, hides the complexities of transistor physics, with the expected semantics that the sum of the operands will be stored in the target register, and that nothing else would change. This interface is vital for the blossoming of the ecosystem of programming abstractions, microarchitectures, software libraries, run-time systems, and applications.

Can we find a generic interface for programming cyberphysical systems? Unlike packaged ICs, cyberphysical systems are *open* and exposed to the environment. The torque generated by a "control instruction" depends not only on the the physics of the motor—which can be considered to be part of the machine—but also on the wildly varying drag. Consequently, every control instruction (controller) is specifically designed for the underlying hardware, the anticipated environment, and the higher-level objectives. Control programs cannot be safely ported from this year's car model to next year's, let alone across different cars or vehicles. This specificity also stymies innovation in higher-level programming abstractions.

Consider programming an advanced alerting system that estimates the space occupied by potentially offending vehicles based on information received from other vehicles as well as motion estimates computed from telemetry and cameras [9, 11]. Here, the notion of the occupied space should not just be a variable that is updated only when new messages are received, but as the communication links and on-board computers can fail, this variable should be updated continuously even in the absence of messages. This calls for a new programming abstractions—perhaps, one with continuous, hybrid, or time-triggered variable updates [6, 7]. Distributed and asynchronous nature of vehicle-to-vehicle and vehicle-to-infrastructure systems bring to mind other kinds of useful programming abstractions [8, 5].

Benefits of a good programming abstraction go beyond improving productivity and portability. As the static and dynamic analysis techniques for cyberphysical systems come of age (see recent developments, for example in [4, 10, 3, 1, 2]), the common language and its idioms can catalyze research on development of significant software frameworks and tools for design automation, verification, and testing.

There are several interesting technical challenges in designing programming interfaces for CPS. For example, their openness require that the CPS interfaces should not only specify progress—what changes with the successful completion of a control instruction, but like the ISAs, they should also specify invariants—things that remain unchanged. These invariants should be useful for compositionally restricting behavior while they should not be so restrictive as to become platform specific or unimplementable. One way forward is be to propose candidate abstractions and demonstrate their utility by building increasingly more general classes of CPS. It should be possible to support and sustain such endeavors as the computing industry has repeatedly demonstrated the profitability of investing in common core abstractions and supporting technologies, while competing for higher-level services and applications.

References

[1] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *CAV*, pages 258–263, 2013.

[2] P. S. Duggirala, T. T. Johnson, A. Zimmerman, and S. Mitra. Static and dynamic analysis of timed distributed traces. In *RTSS*, pages 173–182, 2012.

[3] P. S. Duggirala, S. Mitra, and M. Viswanathan. Verification of annotated models from executions. In *EMSOFT*, pages 1–10. IEEE, 2013.

[4] G. Frehse, C. L. Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. Spaceex: Scalable verification of hybrid systems. In G. Gopalakrishnan and S. Qadeer, editors, *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 379–395. Springer, 2011.

[5] S. Gilbert, N. Lynch, S. Mitra, and T. Nolte. Self-stabilizing robot formations over unreliable networks. *Special Issue on Self-adaptive and Self-organizing Wireless Networking Systems of ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(3):1–29, 2009.

[6] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Giotto: A time-triggered language for embedded programming. In T. A. Henzinger and C. M. Kirsch, editors, *EMSOFT*, volume 2211 of *Lecture Notes in Computer Science*, pages 166–184. Springer, 2001.

[7] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. *The Theory of Timed I/O Automata*. Synthesis Lectures on Computer Science. Morgan Claypool, November 2005. Also available as Technical Report MIT-LCS-TR-917.

[8] H. Kowshik, D. Caveney, and P. R. Kumar. Provable systemwide safety in intelligent intersections. *IEEE T. Vehicular Technology*, 60(3):804–818, 2011.

[9] R. Mangharam, R. Rajkumar, M. Hamilton, P. Mudalige, and F. Bai. Bounded-latency alerts in vehicular networks. In *2007 Mobile Networking for Vehicular Environments*, pages 55–60. IEEE, 2007.

[10] A. Platzer. Differential logic for reasoning about hybrid systems. In A. Bemporad, A. Bicchi, and G. C. Buttazzo, editors, *HSCC*, volume 4416 of *Lecture Notes in Computer Science*, pages 746–749. Springer, 2007.

[11] T. Wongpiromsarn, S. Mitra, A. Lamperski, and R. Murray. Verification of periodically controlled hybrid systems: Application to an autonomous vehicle. *Special Issue of the ACM Transactions on Embedded Computing Systems (TECS)*, 11(S2), 2012.